CLI (command line interface)

# INTRODUCTION

Earlier, there was no UI for devices hence commands had to be given in order to perform any activity on the machine.

Now many devices doesn’t need GUI since there is no need for user interaction (Ex: server, backend device etc) hence CLI is still in use, over 90% of servers in internet run on Linux as it is fast, secure, free and also Non-GUI.

MobaXterm 🡪 is a CLI terminal emulator used to login to any Non-GUI systems using many protocols such as SSH, HTTP, FTP, SFTP, Telnet, Rsh, RDP (Remote Desktop Control), VNC (virtual network computing), Mosh (advanced SSH with added features).

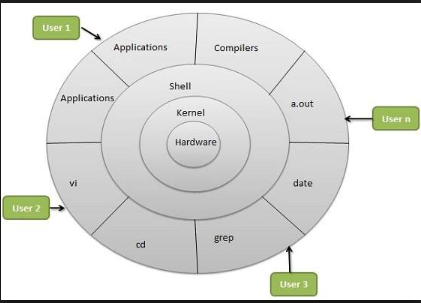
Unix 🡪 developed by Bell lab, owned by them.

Linux 🡪 Linux is a free, open source OS using linux kernel.

* Linux is an UNIX clone, but developed by Linus Torvals from scratch.
* Since it is open source, libraries can be edited /added newly and redistributed. Thus each redistribution (distros) can be a new linux OS, for example:
  + Red hat Enterprise Linux
  + Ubuntu
  + Linux Mint
  + Android
  + Debian
  + Fedora
* Provides Virtual Terminals for concurrent usage/multitasking (i.e each user/instance can be accessed without disturbing other user) thus,

**$ Ctrl+alt+ F1** 🡪 logged into VT\_1.

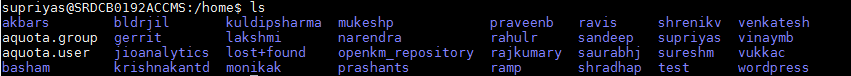
* **Worked on Linux distros Ubuntu 3.2.0**



Linux File structure: Root directory (cd /)



Users inside home directory:



Shell 🡪 A software program that accepts commands from User (in turn Terminal), processes it and shares the result back on the terminal.

Thus, shell is a command language interpreter.

Terminal 🡪 A program that runs shell. (Mainly a UI interface between Shell and user)

Kernel 🡪 Connection between hardware and software/os of the machine.

## **Types of Shell**

1. Bourne Shell (sh)
   1. Uses $ as prompt
   2. Suitable for scripting, programming.
2. C shell / Csh
   1. Uses % as prompt
   2. More user friendly (c based scripting commands), interactive usage (UI).

* Korn Shell (ksh)

Developed/based on sh source code, a sandwich of sh & csh.

* Bash (Bourne again shell)

Incorporates feature from sh, csh, ksh (used in Linux, Unix OS)

## **Difference between LINUX vs UNIX**

|  |  |
| --- | --- |
| LINUX | UNIX |
| Open Source | Priced ownership |
| CLI + GUI | Only CLI (developing GUI) |
| Flexible and Compatible with all hardware | Less compatible with different hardware |
| Used in Mobile, PC, Desktop etc | Used in Server, High end computers |
| Distros: Ubuntu, Red hat, Linux Mint, Solaris | Distros: BSD, IRIS, AIS |

# **Shell COMMANDS**

|  |  |
| --- | --- |
| **Command** | **Description with usage** |
| #  $  Ctrl + C  $@ or $\*  $? | Used for commenting in shell scripting.  Used to signify commands in documentations.  **Stop any command running in shell (very important)**  Holds list of all arguments passed to the script  Returns exit code i.e error codes if any. |
| sudo shutdown –h now  sudo shutdown hh:mm  sudo shutdown +mm  sudo shutdown –h hh:mm  sudo shutdown –p hh:mm  sudo shutdown -c  sudo halt | Machine/System shuts down at the moment.  System shuts down at mentioned time.  System shuts down after minutes specifies.  Ex: $ sudo shutdown +30 🡪 shut down after 30 minutes.  System shuts (stops all the processes) @ specified hour and minute.  Ex: $ sudo shutdown –H 1:20 🡪 system halts all processes @ time = 1.20  System halts processes and shuts down @ specified hour and minute.\  To stop shut down, type this command in different terminal. |
| sudo shutdown –r /  sudo reboot | To restart/reboot the system use this command. |
| sudo pm-suspend  sudo pm-hibernate | Suspends the system  (i.e. saves current state, almost shuts down, still consumes the power/electricity)  Hibernates the system (i.e. saves current state, powers off the machine) |
| shutdown [option] [time] | Option to shut down (-h)/reboot (-r) the system.  Ex: $ shutdown –h now ; s/m shutdown now  $ shutdown –h +10; s/m shutdown after 10 minutes  $ shutdown –r now; s/m reboot now |
| Sleep [time in seconds] | Sleep for mentioned seconds  Ex: $ sleep 10 ; 🡪 system waits for 10 seconds |
| Wait [pid] [jid] | Wait until the mentioned process ID or Job ID is terminated.  Ex: wait 2017 🡪 wait until 2017 process ID terminates and exit it’s status. |
| [cmd] –-help /  man help [cmd]/man [cmd] | To Display information of any command.  Example: $ ls –-help; 🡪 Displays information about ls command options.  $ man help touch; 🡪 Displays touch options (press ‘q’ to quit)  $ man mkdir; 🡪 Displays usage of mkdir command. |
| help | Displays usage of all the supported commands of CLI.  Some commands differ in CLI as different versions of Linux/unix supports different command usages. |
| Glob using Wildcards  \*  ?  [ ]  ! | Many symbols can be used during pattern matching or globbing for files/contents of file, these symbols are called wildcards.  Ex: hello\* 🡪 signifies any number of characters after hello word.  Ex: hello? 🡪 Signifies single character after hello word.  Ex: hello[i-k] 🡪 signifies characters after hello word from I to k.  Ex: hello[!3] 🡪 signifies to ignore single character 3 after hello word. |
| Apropos [option]  whereis [command]  Locate [file]  Find [ ]  Tab key | Used in Unix s/m to **search a word in the command’s description**.  Ex: $ apropos edit;   * All the commands containing edit in their description is displayed.   Ex: $ apropos copy paste;   * All commands containing copy/paste in their description is shown.   To check if/where a **particular command exists.**  Ex: $ whereis ls; 🡪 o/p = C:\Program Files\Git\user\bin\ls.exe  Find path of the file.  Ex: $ locate –i hello.txt 🡪 displays all file paths having hello.txt file.  Find files/directory with their absolute path.  Ex: $ find –name sample\* 🡪 find all file starting with sample in the directory.  Tab key can be used to display all the available files in the system.  $ press Tab twice 🡪 all the files will be displayed.  $ c + press Tab; 🡪 all available files starting from “C” will be displayed. |
| Less  more  Grep  Sed (stream editor)  Awk  Spell  Wc (word count)  Pg  Uniq  Read  IFS | Display contents of the file, one page at a time. Scroll for next pages.  Ex: $ less sample.txt  Display entire contents of file, scroll over.  Ex: $ more sample.txt  Search exact word/data within/across all files.  Ex: grep “ta” file.txt 🡪 search word ‘ta’ in file.txt.  Search, find, replace, insert, delete.  s 🡪 substitution option, used for replacing.  g 🡪 select all occurrences  1,2,3 🡪 select 1st, 2nd, 3rd … occurrence.  / 🡪 delimiter  Ex: $ sed ‘s/unix/linux/’ sample.txt 🡪 replace unix with linux in sample.txt  Special program, used for searching data in files, tables (rows, columns) as well.  Ex: $ awk ‘/manager/ {print}’ sample.txt 🡪 search manager in sample.txt file.  To check misspelled words of a file with dictionary.  Ex: $ spell –on file1.txt file2.docx  🡪 checks and displays the misspelled words of files, file1 and file2  Displays number of lines, word count, bytes count in the file.  Can be used only on file not on folder/directory  $ wc file.txt ; 🡪 o/p = 12 390 330 file.txt  So there are 12 lines, 390 words of 330 bytes data in file.txt  Display text file one page at a time (just like more, less command)  Ex: $ cat file.txt | pg -3  🡪 displays first 3 lines of file.txt  Show **occurrence of Line**, delete repeated lines from input file and display result to output file.  Ex: $ uniq –c sample.txt 🡪 displays unique count of each line in sample.txt  Used to read data from **keyboard/user,** also to split data into words.  -r 🡪 do not allow backslash to escape characters.  -n 🡪 limit number of input lines  -p 🡪 prompt message, displayed on the console of same line  Ex 1: $ read –n 5 -p “Enter 5 character only”   * Enter 5 character on console, output will contain entered characters.   **Internal Field Separator,** used with read for word splitting/line splitting.  Ex 2: create file.txt, with contents = hi hello welcome  Script : file=file.txt  IFS=’ ‘ # space is used as IFS to separate words  While read –p “input 3 IDs” id1 id2 id3  Do  Printf “print id1 $id1”  Printf “print id2 $id2”  Printf “print id3 $id3”  Done < “$file” # first line to be executed, redirect file data. |
| cksum file\_name | Unique ID/Identifier assigned for every file, to preserve data integrity. |
| pwd | Displays path of the working directory |
| **<**  **>** | Input redirection from console to file/vice versa.  Ex: mail –s “Welcome Email” [receiver@mail.com](mailto:receiver@mail.com)< file.png  Output redirection from console to file/vice versa.  Ex: echo “hi hello”>file.txt 🡪 override/redirect the data to be added into file.txt  Ex: echo “welcome”>>file.txt 🡪 without overriding existing data, file will be edited by adding the entered input at the last.  Ex: find sample 2 >error.log 🡪 used to find a file named sample, if error then redirect error details to error.log file (done by ‘2’ in command) |
| echo XX | Prints data on console/terminal.  $ echo Hi\n 🡪 Hi\n is displayed on console  $ echo hello! Its me>>file.txt 🡪 add data (hello! Its me) into the file.txt |
| printf XX | Same as Echo but it considers \n as line break.  Ex: $ printf “hey \n hello” 🡪 output will be as below:  hey  hello |
| history | Displays all the commands ran in the s/m (commands used from beginning).  $ history; 🡪 sequentially displays all the commands executed by user. |
| Ctrl + R | To search from entire shell history.  $ cltr+R; 🡪 Displays reverse-i-search.  After ctrl+R type in the command you were searching, latest one is displayed. |
| l or ls | Lists all the files |
| $ ls ; 🡪 displays all files in row.  $ ls -1 🡪 display all files in column.  $ ls \*.jpg ; 🡪 displays all files ending with .jpg  $ ls JPG\*; 🡪 displays all files starting with JPG.  $ ls \*(jpg, png, mpeg); 🡪 displays all files ending with jpg/png/mpeg.  $ ls \*(jpg, png); == ls \*jpg, \*png ; == ls \*{jp,pn}g ; (all display same o/p)  $ ls j?g; 🡪 Displays file starting with ‘j’, ending with ‘g’ any single character in between.  $ ls –a; (/–all/-ah) 🡪 displays all files, hidden files as well.  $ ls –lh; 🡪 displays files with their size, creation date, time, path of file.  $ ls –s; 🡪 displays file with their size  $ ls –lah; 🡪 displays all files (hidden as well) with their sizes.  $ ls –R; 🡪 shows all files in directory + their subdirectories.  $ ls –l 🡪 display all files/directory with their permissions. |
| lsof | Lists all the open files  $ lsof 🡪 displays all the files open |
| Ls -l | Lists permission available to user on all the available files/directory.  It’s divided in 3 :(-xxxyyyzzz) or (dxxxyyyzzz) where -/d is for file/directory.  xxx 🡪 permission level for user  yyy 🡪 permission level for user group  zzz 🡪 permission level for others  Ex: $ ls –l;  If o/p 🡪 -rw-rwx-wx : first - 🡪 indicates it’s a file.  If o/p 🡪 drwx-wx-w- : first d 🡪 indicates it’s a directory |
| Conditions on naming a File/directory:   * It can contain any character but cannot have ‘/’ * If name starts with any special character (such as #$% etc) put the name in ‘ ‘ (called as quoting) or precede each special character with ‘/’ (called as escaping).   Ex: ?$file == ‘?$file’ == /?/$file | |
| mkdir XXX | To create a folder/directory. |
| $ Mkdir file\_name; 🡪creates a directory file\_name;  $ mkdir a.txt ; 🡪 creates a folder with name “a.txt” not a txt file.  So to create a file with proper format use “touch” command. |
| dir | Lists only the directories in the path.  Ex: $ dir 🡪 list directories available in the current path. |
| touch XXX | To create any file.  Ex: $ Touch a.txt; 🡪 creates a text file |
| rmdir XXX  rm XXX  rm [option] XXX | To remove **any empty** folder/directory.  Ex: $ rmdir file\_name;  To Remove any empty file;  Ex: $ rm file.txt 🡪 removes individual file from the directory.  Removes the file/folder specified with some specifications.  -r, -R, --recursive: Remove directories and their contents recursively.  -f: Ignore nonexistent files, never prompt  -i : Prompt before every removal.  -- version: Display version information, and exit.  -v: verbose, to explain what is being done.  Ex: $ rm –r file4 ; 🡪 deletes all the contents of file4.  Ex: $ rm –rf directory 🡪 deletes the folder and all of its contents. |
| cp [src] [dest] | Copies file from source to destination.  $ cp C:/user/file backup; copies all files from source to backup folder |
| mv [src] [dest] | To move/rename files from source to destination.  $ mv file1.txt folder4 ; 🡪 moves file file1.txt to folder4;  $ mv file1.txt file9.txt 🡪 renames file1 as file9;  $ mv file.txt file.docx 🡪 changes file format from txt to docx; |
| diff fileX fileY | Compares filex with filey and displays all the mismatched lines b/w files.  Can only compare b/w files of same format (i.e., extension)  $ diff File1.txt File2.txt ; |
| Consider below file structure for ‘cd’ command reference:  (parent directory) Home 🡪 Supriya (base directory) 🡪 file1 🡪document2🡪 w.txt;  🡪 user1 🡪 project1, e.txt, hello.docx  🡪 user 2 🡪 welcome, how.pdf, project2 | |
| cd | Change directory,  move to base/working directory of user from within user paths.  Ex: your CD = document2; 🡪 $ cd ; 🡪 your CD = Supriya;  Ex: your CD= home; 🡪 $ cd ; 🡪 your CD = Supriya (base directory) |
| cd [path] | Move to the path specified only if it exists.  Your CD = file1; 🡪 $ cd document2; 🡪 moves to document2 folder.  Your CD = project1; 🡪 $ cd p ; 🡪 ERROR, no file found  Your CD= home; 🡪 $ cd Supriya/file1/document2/; 🡪 moves to document2 |
| cd .. | Move step -1 directory  Ex: your CD = project1; 🡪 $ cd .. ; 🡪 your CD = user1;  Ex: your CD = user2 ; 🡪 $ cd .. ; 🡪 your CD = home;  cd .. = cd /home  cd ../Supriya = cd /home/Supriya |
| cd - | To switch between last 2 recently used directories. |
| cd / | From anywhere to base/root directory.  Your CD = project1; cd / 🡪 your CD = Home |
| clear | Clears the CLI of all activity/commands. |
| Exit/ Ctrl + D | To logout from the machine |
| quit | To exit the terminal window without closing shell. |
| ‘ “ ( ) | If mistakenly any special character is typed in shell prompt, to get back shell prompt type same characters closure as shown below.  Example1: If ‘ is typed; type ‘ again to get back shell ;  Example2: If ( is typed; type ) again to get back shell ; |
| Date  Cal | Displays date and time w.r.t country  Example: $ date ; displays Thu Sep 28 09:00:01 IST 2017  Displays this month’s calendar |
| Hostname [option] | $ hostname 🡪 Displays the name of the host i.e., name logged in as.  $ hostname –I 🡪 Displays IP address of host |
| Users | Displays the name of the user logged in as. |
| uname [option] | Prints kernel name/hostname/kernel release number of the s/m.  $ uname ; o/p = Linux 🡪 meaning we are running on linux s/m.  $ uname –a; 🡪 complete linux system information is displayed  $ uname –r 🡪 kernel information is displayed. |
| bash --version | Gives the version, copyright, license details of bash. |
| Cat [option] | $ cat /proc/cpuinfo 🡪 Displays CPU information  $ cat /proc/meminro 🡪 Displays Memory information |
| df –h/free | Prints memory (total, used, free space) details of the server/home.  $ df –h ; or $ free |
| unzip xxx  unzip –l xxx | To unzip a compressed zip file.  To display contents of zip file without unzipping it. |
| cat [xxx] | Used to create, display, copy data from files.   1. Create a new file   $ cat>file1.txt ; 🡪 creates file1.txt, opens file to start adding data.  Now, Enter whatever data to be entered within file.  $ ctrl+D to close adding data.   1. Create a new file and add contents from old to new file.   $ cat file1.txt > file4.docx; 🡪 Creates a new file, file4.docx if not already present then copies data from file1.txt to file4.docx.   1. Displays contents of a file.   $ cat file.txt ; 🡪 displays content of file.txt  $ cat file1.txt file2.html file3.docx ; 🡪 displays content of all 3 files.   1. Copy and paste complete content of one file to another file.   $ cat file1.txt > file2.html ; 🡪 entire data of file2 will be deleted and pasted with file1 data. |
| Head [xxx] | Used to display file/file data in ascending order/display first few lines.  Ex: $ head file.txt 🡪 displays content of file.txt  Ex: $ head -3 file.txt 🡪 limit the file display to only 3 lines from first. |
| Tail [xxx] | Used to display file data in descending order/to display last few lines of file.  Ex: $ tail file.txt 🡪 displays content of file.txt  Ex: tail -3 file.txt 🡪 limit the file display to only 3 lines from last. |
| Tree [xxx] | List contents of directory in tree format.  Ex: $ tree folder 🡪 lists content of folder in tree format.  Ex: $ tree 🡪 go into folder for which tree is required, then type tree command. |
| expr [xxx] | Expression commands to do any arithmetic/logical function.  $ expr 2 + 2 ; displays o/p = 4  $ expr 2 = 6 ; displays o/p = 0 cause the expression is untrue/false |
| bc | Opens Basic calculator terminal with display of license, copyright information of the software used. Can do all the arithmetic operations.  Ex: $ bc🡪 terminal opens.  Type pi = 10; r=70; a= 100;  Now type pi+r-100 🡪 o/p = -20  To come out of window use $ quit. |
| uptime | Displays the machine uptime/run time in detail (login time, users, load etc)  EX: $ uptime;🡪o/p = 10:00:00 up 1 day,10 min,2 user, load average: 0.01, 0.1, 1  The format and data could differ between machines. |
| vim/ less/ nano/ joe/ emacs | Vim 🡪 is a “vi improved” text editor used to edit/modify a file through CLI.  To login and edit the file using terminal (file content is displayed on terminal).   1. $ vi try\_vi🡪 creates file “try\_vi”, then opens the terminal for usage. 2. $ press ‘ESC’ then ‘I’ 🡪 to enter INSERT mode to edit/write in file. 3. $ press ‘ESC’ then :w 🡪 to save the edited/modified file 4. $ press ‘ESC’ then :wq 🡪 to save file and quit the editor. |
| start \\IPaddress  stop/restart/reload/status | To connect to any device (laptop) using its IP address and access it’s data.  It can also be used to stop, restart, reload and find status of connection.  Ex: $ start [\\100.8.19.2](file:///\\100.8.19.2) ; $ stop [\\100.8.19.2](file:///\\100.8.19.2) |
| Netstat | (Network Statistics) Displays information of protocols in use, helps in monitoring incoming and outgoing connections to/from the machine.  Ex: $ netstat; 🡪 displays protocol, send, receive, address etc   * Keeps running until it is stopped with Ctrl+C |
| Ping | To Test a connection request between a host and server (uses ICMP protocol)  Ex: $ ping 8.8.8.8 8888🡪 pings IP address with port  And displays connection request time, packet and data stats.  Ex: $ ping –c3 8.8.8.8 🡪 pings Google’s DNS server 3 times.  Ex: $ ping yahoo.com 🡪 pings yahoo and returns it’s IP with data msgs.   * Keeps running until it is stopped with Ctrl+C |
| SSH XXX | SSH: Network protocol used for communicating securely over a non-secure connection.  $ SSH xx.xx.x.x; 🡪remote login to the given IP address |
| FTP XXX | FTP: Network protocol used for file transfer b/w client and server over a computer network.  $ FTP xx.xx.x.x 🡪 To login to remote FTP location |
| SFTP XXX | SFTP: Network protocol used for file transfer b/w client and server over a secured port/connection. (SFTP = SSH + FTP)  $ SFTP xx.xx.x.x 🡪 To login to remote SFTP location |
| nslookup | Shows IP address of hostname and the serving DNS server.  Ex: $ nslookup [www.google.com](http://www.google.com)   * Server: xx.xx.x.xxx 🡪 *this is DNS server address* * Address: xx.xx.xxx.x#xx 🡪 *host’s address*   Ex: $ nslookup localhost   * Server: 127.0.0.1 * Address: 127.0.0.1#xx 🡪 #xx is the subnet IP allocated within a group. |
| host [URL/IP] | Provides information of the URL hosted.  Ex: $ host [google.co.in](http://www.google.co.in)   * Shows IP addresses hosted by google   Ex: $ host xx.yy.ahs.aj   * prints host name of site etc. |
| Telnet | A Network protocol used to connect to a remote host over unsafe connection.  $ Telnet xx.xx.x.x 🡪 Connects to the IP address remotely. |
| ipconfig [XXX] | To display IP configuration of the machine only if it is windows OS.   * displays all current TCP/IP network configuration values * refreshes Dynamic Host Configuration Protocol (DHCP) * refreshes Domain Name System (DNS) settings   Ex: $ ipconfig 🡪 displays IPv4 address etc,  $ ipconfig /all 🡪 displays detailed n/w information.  $ ipconfig /renew 🡪 renews all adapters |
| netcat  nc xx <host><port> OR  ncat xx <host><port> | Netcat is used for any random TCP and UDP connections and listeners.  It can also be used for HTTP connection with the help of $ printf.   * It can open any TCP/UDP connection * Send UDP packets to server or receive from client. * Can listen/connent to any TCP/UDP ports * Deal with both IPv4 and IPv6, do port scanning. * $ Ctrl+D 🡪 command to close the nc Session   Ex: 1. TCP Client – Server Model (between 2 different machines)  On client (machine 1) 🡪 $ nc –l 1234 🡪 nc is listening on port 1234  On Server 🡪 (machine 2) 🡪 $ nc 127.0.0.1 1234 🡪 nc connects client and server at port 1234 successfully. (127.0.0.1 is IP of  Ex: 2. TCP Client – Server Model (between same machine)  Open 2 tabs on same machine and try below commands on the tabs:  $ nc –l 2046 🡪 nc is listening on port 1234  $ nc localhost 2046 🡪 connection established at port 2048 on same machine.  Ex: 3. Using nc with printf to connect to server using HTTP  $ nc printf 'HEAD / HTTP/1.1\r\nHOST: www.google.co.in\r\n\r\n' | nc www.google.co.in 2046   * This helps to connect to Google using HTTP request and HTTP response will be displayed on the terminal.   Refer to manual page for more details ($ man help nc) |
| traceroute [domain\_name] | Used to display route/hops for traffic to reach client from server.  Ex: $ traceroute google.com 🡪 displays no of hops for google to send data back to host or vice versa. |
| tcpdump [options] | Prints out contents of packet transfer on network interface.  Need special privilege to use this tcpdump command.  Ex: $ sudo tcpdump –n host 8.8.8.8;   * Ping query, Shows IP address of host, the packet transfer between host and 8.8.8.8 (which is google’s server), request and reply time etc.,   Ex: $ sudo tcpdump –n port 53;   * DNS Query |
| wget [path of file] | Use Wget command to download files on a linux machine.  Ex: $ wget <http://www.google.com/folder1/file.zip>  Downloads above zip file in the linux machine. |
| **ps ux** | Displays status of all the processes running under a user.  $ ps 🡪 display current running processes of the user.  $ ps –ef 🡪 display currently running process across system.  $ ps ux 🡪 displays, user, PID, %cpu, %mem, start time, command, etc. |
| Kill  Kill PID [id value] | Used to Kill/close any process.  $ kill PID xxx 🡪 kills a process having processID (PID) xxx. |
| mail -s 'subject' to-address | To send Email to Sender.  $ mail –s “Welcome” [sender@mail.com](mailto:sender@mail.com); |
| mail -s 'sub' -c 'cc’ –b 'bcc' to-address | mail -s 'subject' -c 'cc-address' -b 'bcc-address' to-address;  To send Emails to Sender with cc, bcc list.  $ mail –s “Welcome!” –c [abc@mail.com](mailto:abc@mail.com) –b [def@kmail.com](mailto:def@kmail.com) [receiver@mail.com](mailto:receiver@mail.com) ; |
| mail -s "Subject" to-address < Filename | To send email with Attachment.  $ mail –s “Welcome Email” [receiver@mail.com](mailto:receiver@mail.com)< file.png |
| tar[option][file][path]  -c  v  f  z  j  -x  -t  -r | Tar file 🡪 highly compressed files (compressing file size level: tar> gzip> bzip)  Creates a new tar archive file  Show tar file progress  File name type of archive file  Creates a gzip archive file  Creates a bz archive file (format = bz2 or tb2 or tbz)  $ tar –cvf f1.tar /home/folder1 🡪 f1.tar file will be created under folder1.  $ tar cvf**z** f2.tar.gz /home/folder2 🡪 creates compressed gzip file f2.tar  $ tar cvf**j** f3.tar.bz2 /home/folder3 🡪 creates highly compressed bz file f3.tar  To Untar/extract the .tar files use “**x**” in command  $ tar –xvf file1.tar /sample 🡪 unzips file1.tar in sample folder  $ tar -xvf fil2.tar.gz 🡪 unzips gzip file  $ tar -xvf fil3.tar.tbz 🡪 unzips bzip file  To Display contents of Tar file  $ tar –tvf file1.tar 🡪 all files under file1.tar will be displayed  $ tar –tvf file2.tar.gz 🡪 lists file under gzip  $ tar –tvf file3.tar.tbz 🡪 lists file under bzip  Adding files to tar file  $ tar –rvf file1.tar sample.txt 🡪 sample.txt file will be added to file1.tar  $ tar –rvf file2.tar.gz sample.txt  $ tar –rvf file3.tar.tbz sample.txt |
| Pipe [$ | $ | ……] | Pipe helps to take output of 1st command and serve it as input to 2nd command. Just like a physical pipe, hence the name.  Ex: $ cat file.txt | less 🡪 displays only one scroll length of file data of file.txt  Ex: $ ls -1 | sort –o file.txt | |
| Sort [options] | Used to sort the files, text data in files alphabetically.  -r 🡪 display in reverse order  -n 🡪 numeric sort  -o 🡪 write sort output to a file  Ex: $ sort file.txt 🡪 sorts contents of file.txt alphabetically.  Ex: $ ls -1 | sort –o file.txt | ls -1 🡪 display, write sort output to file.txt, display |
| Export [options] | To export/output the environment variables  -p 🡪 list names exported in current shell  -n 🡪 remove names from export list  -f 🡪 export name as functions  Ex: $ export –p 🡪 displays all variables used in current shell. |
| **Ownership**  **In Linux every user is assigned an account to store their data, there can be multiple users.**  **User**: User/owner who creates a file and has all the permissions to it.  **Group**: Group of people having access/permission to a set of file/folder.  **Other:** Person who hasn’t created file/folder and doesn’t belong to any group. | |
| **Permissions**  **Read**: Ability to open, read and list contents of a file.  **Write**: Authority to add, remove, rename files in a directory, but cannot rename, move, delete the directory.  **Execute**: Authority to run, see and modify a .exe file (if no permission cannot execute/run .exe file)  r-read; w: write; e: execute; - : no permission; | |
| sudo [option] [user]  [options]  adduser  userdel  passwd  groupmod  usermod  deluser  finger  lsof  chown user  chgrp group\_name filename | Permits a user to execute a command as super user/admin.  Options that can be done with sudo command.  $ sudo adduser user1 🡪 creates a new user user1, enter password etc.  $ sudo userdel –r user1 🡪 deletes a user (here user1)  $ sudo passwd –l user2 🡪 deletes password of user2 & disables account.  $ groupmod “press Tab key thrice” 🡪 displays all the groups on the machine  $ sudo usermod –a –G [group name] [user name];  Ex: $ sudo usermod –a –G groupmain user10 ; 🡪 add user10 to groupmain  $ sudo deluser [user name] [group name];  Ex: $ sudo deluser user10 groupmain;  Gives information on logged users in local and remote machine.  Ex: $ finger  $ sudo lsof –I 🡪 lists all the open files running on ports.  Changing ownership to file/directory.  Ex: $ sudo chown user1 folder1 🡪 change owner of folder1 to user1.  Ex: $ sudo chown user9 folder2 🡪 change owner of folder2 to user9.  Changing ownership of group owner.  Ex: sudo chgrp root test1 🡪 owner of group test1 changed to root. |
| chmod | Permission level can be changed using chmod command in 2 ways:   1. Absolute mode (using numbers in command, change user/group/other level)   *Note: remember Binary level coding*   |  |  |  | | --- | --- | --- | | Number | Permission | Symbol | | 0 | No permission | --- | | 1 | Only execute | --x | | 2 | Only write | -w- | | 3 | Execute + write | -wx | | 4 | Only read | r-- | | 5 | Read + Execute | r-x | | 6 | Read + Write | rw- | | 7 | Read + Write+ Execute | rwx |   Ex: $ chmod 764 sample 🡪 change permission of sample folder to drwxrw-r—.  Ex: $ chmod 755 file1.txt 🡪 change permission of file1.txt to drwxrw-r—.  i.e. rwx access to owner; rw access to user group; only r—access to others.   1. Symbolic mode (using symbols in command, change individual owner level)  |  |  | | --- | --- | | Operator | Description | | + | Add permission to file/directory | | - | Delete permission to file/directory | | = | Set permission by overriding previous permission |   u 🡪 user/owner  g 🡪 group/user-group  o 🡪 other  a 🡪 all  Ex: $ chmod o=rwx sample  i.e. Set permission to sample folder, for others as read, write and execute.  Ex: $ chmod a=r—sample.txt  i.e. set permission to file sample.txt with only read access to all. |

# Shell Scripting

Shell scripting is writing a series of command for the shell to execute. It can combine lengthy and repetitive sequences of commands into a single and simple script, which can be stored and executed anytime. This reduces the effort required by the end user.

* Commands run in single line & perform activity.
* Commands are Case sensitive.

## Steps in creating a Shell Script:

1. **Create a script file** **using** the **vi** editor i.e. vim (or any other editor)

🡪 **$ vi filename.sh**

Above command opens the file in vim editor, 🡪 ESC 🡪 I for insert mode.

1. **Start** the script file with

**#!/bin/bash or #!/usr/bin/env bash**🡪 (Shebang) to direct the script to run on bourne/bash shell only. It should be the first line of the script, if not it gets ignored as comment.

**ESC🡪:wq 🡪** saves the file and exists vim editor.

1. To Execute the script

*$* ***bash filename.sh* or *./filename.sh***

(If execute permission is provided already, if not use ***$ chmod 7xx filename.*sh** to give execution permission of the file to owner/creater).

* **Shebang** 
  + #!/usr/bin/env python - for python scripts
  + #!/usr/bin/env node – to run javascript
* Refer: <http://www.sanfoundry.com/1000-linux-command-tutorials/> for all shell commands.
* Refer <https://www.guru99.com/communication-in-linux.html> for quick guide.
* Refer <https://www.guru99.com/install-linux.html> for Virtualbox+Ubuntu installation on PC.

## Variables

* Variables are defined is key=value pair (without spaces between key and value).
* Starts with alphabet/underscore (never with integer/number), case sensitive.
* Can use alphabet, underscore, and numbers.
* If key or value has spaces, enclose it within ““. Thus no spaces are allowed anywhere.
* $ Symbol is used refer to variables.
* -z 🡪 used to check if variable is empty/not (ex: if (-z $var) i.e if var is empty)

Example script:

#!/usr/bin/env bash

Name=bob

“Person age”=”10 years”

Echo $name is $person age years old.

## Parameters (0 to 255)

* $0 – name of the script
* $1 – first parameter – upto - $9 – ninth parameter
* ${10} tenth parameter - to - ${255} – final parameter
* $@ - continuous fetching of data from the user
* Echo $? 🡪 returns exit code (from 0 i.e success to 255 i.e if any errors in script)
* Use ( ) for enclosing command within scripts such as date, pwd, cat etc

1. Example script: echo data from the user for 1st parameter

*#!/usr/bin/env bash*

*User =$1*

*Echo Hello $user*

* Execute $ bash file.sh Joe 🡪 Output = Hello Joe

1. Example script: echo data from the user for 5th parameter

*User =$5*

*Echo Hello $user*

* Execute $ bash file.sh Joe Mi Ki Lo Po 🡪 Output = Hello Po

1. Example script: echo predefined parameter date

*Echo Hello time is $(date)*

* Execute $ bash file.sh 🡪 Output = Hello time is Tue Feb 5 10:00:00 IST 2019

1. Example script: echo predefined parameter date

*Echo Hello, create file $(touch new.sh)*

* Execute $ bash file.sh 🡪

Output = Hello, create file 🡪 check in directory new.sh will be created.

## Operators

* +, -, \*, / , %, ==, !=, >, <, >>, <<, ++, --, exp,
* -eq, -ne, -le, -lt, -gt, -ge
* Files related: -r, -w, -e, -s, -e

## Conditions

* Always give spaces between operators and variables used within condition.
* Use [ ] for conditions
* *IF THEN FI* condition
  + if [ condition ]

then

--body when true--

fi

Example: if [ $color = ”blue” ]

then

echo “condition is true”

fi

* *IF THEN ELSE FI* condition
  + if [ condition ]

then

--body when true--

Else -- body when false--

fi

Example: if [ $color=”blue”]

then

echo “condition is true”

else echo “condition is false”

fi

* *IF THEN ELIF THEN ELSE FI* condition
  + if [condition ]

then

--body when true--

elseif [condition]

then -- body when untrue—

else –body when false --

fi

* *While* condition
  + *While [ condition ]*

*do*

*--body if true--*

*done*

* *For* condition
  + For ( ) in ( )

do

-- if body is true—

done

Example:

* + for (( c=1; c<=5; c++ )) /
  + for (( ; ; )) /
  + for A in 1 2 3 4 5 / for A in { 1 .. 5} /
  + for file in $FILES /
* *Break* and *Continue*

## Functions

* Functions are used to define actions, run group of instructions as one.
* Use backtick ` ` to enclose commands that should run before main command in the script, also to run the complete command as one and share output.
* Syntax: (with function keyword / without keyword)
  + *function func\_name(){*

*--function body--*

*}*

* + *func\_name(){*

*--function body—*

*}*

Example 1:

* *function get(){*

*local=$1*

*echo “value is $local”*

*}*

*get $1*

* output will be value from user 🡪 ***value is xx***

Example 2:

* *function get1(){*

*FILES=`ls –l | sort –r | head -3`*

*Local=1*

*For file in $FILES*

*Do*

*Echo “File #$local = $file”*

*((count++))*

*}*

*get1*

*Exit 0*

Example 3: read data from file.txt from the script file.sh

* create file.txt - with text data
* create file.sh

*func read\_data(){*

*count=1*

*while IFS=’ ’ read –r var1 #IFS is set to empty/space*

*do*

*echo “var1 $count: $var1”*

*((count++))*

*done < “$1”*

*}*

*read\_data*

*exit 0*

* *run the script : $ ./ file.sh file.txt 🡪 output = contents of file.txt will be displayed*

# OVERVIEW

Help, man, man help,

vi filename, esc I, esc :w, esc :wq,

touch, cat, mkdir, rmdir, rm –r, rm –rf, cd, cd.., cd, pwd, ls (-l,-a), tree, dir

cp, mv,

bash, ./, chmod, #!/usr/bin/env bash,

if then fi, if then else fi, if then elseif then else if,

key=value ($1, $2, $(pwd), $variablename)